M.S. THESIS

TLS Cross Credential (TLS-CC) for Authentication in Delegated Networks

위임된 네트워크에서의 인증을 위한 TLS 상호증명

2017년 8월

서울대학교 대학원

컴퓨터공학부

Pwint Myat Kay Khine

Abstract

TLS Cross Credential (TLS-CC) for Authentication in Delegated Networks

Pwint Myat Kay Khine
Dept. of Computer Science Engineering
The Graduate School
Seoul National University

Nowadays, most of the content providers such as media and entertainment companies use the Content Delivery Network (CDN) services for faster delivery and higher availability. Using a globally distributed server infrastructure to absorb the network traffic, CDNs are believed to offer faster experience to the end-users and a degree of protection from Distributed Denial of Service (DDoS) attacks. However, despite the benefits of such features, there are several drawbacks related to the authentication of the third party edge networks of CDN. Current mechanisms either trust the CDN providers with the private keys or allow a certification authority to issue the CDN a certificate. Both mechanisms are undesirable in terms of attack space expansion due to the sharing of private keys or in terms of domain confusion and complicated

revocation process of the CDN's certificate.

This paper proposes an authentication mechanism in CDN edge networks which does not require trusting the CDN or allowing the certification authority to issue a shared certificate to CDN. Using an object called a "cross credential (CC)" which can prove the delegated relationship between the CDN edge and the origin server, the proposed mechanism offers efficient solution to the above security concerns with extremely low latency and computation overhead compared to the existing solutions. We implemented our proposed mechanism by extending the standard Transport Layer Security (TLS) protocol to create the CC in the back-end channel and verify the CC in the front-end channel for edge server authentication.

Keywords: Content Delivery Network, Transport Layer Security, Delegation, Authentication

Student Number: 2015-23302

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of many web service providers these days is the provision of faster delivery of contents to their end users and the efficient resilience against DDoS attacks. One of the most widely used approaches for achieving this goal is to receive web hosting service from third party hosting providers that have geographically distributed machines around the world. CDN vendors such as CloudFlare, Amazon, and Akamai are the most popular third-party hosting providers these days because they have multiple servers located at the 'edge' of the Internet so that requests from users can be handled efficiently from the edge servers close to them [1].

Since HTTPS performs the end-to-end encryption using TLS protocol as described in [2], the server must authenticate itself in order to establish a secure connection with its client. This is pretty straightforward if the trust model involves only two parties (a client and a web server) in which the server can authenticate itself by providing its own certificate. However, in the case

of getting the web hosting service from a third-party CDN vendor, there has been a delegation problem in authentication which is described in previous work [3].

The authentication mechanisms currently used by the third-party CDN vendors that provide web services on behalf of their customers involve two types of certificates. Imagine Alice, the domain owner of a `www.alice.com` has registered a web hosting service at a CDN vendor, Carol. In order to allow Carol to serve web services instead of his customer, Alice, there are two types of approaches. The first one is to get the private key of Alice so that she can convince the end user who tries to connect to `www.alice.com` [4]. The second one is to issue Carol a certificate which has the domain of Alice in the Subject Alternative Name (SAN) field [5]. The certificate used in the former approach is called a *Custom Certificate* and the one used in the latter approach is called a *Shared Certificate*.

Both of the third party CDN authentication mechanisms mentioned above are problematic in terms of credential sharing which violates the least privilege principle of public key cryptography [6]. The first scheme which transfers the private key to the CDN vendor means delivering all trust to the numerous edge servers distributed around the world. The wider the spread of the shared private key, the more difficult it is to manage the key, and the more likely an attacker can compromise the key. The second scheme which uses a Shared Certificate with the use of SAN field is also problematic since most of the CDN providers use "Cruise-liner" Certificates which include a list of different organizations in the SAN field. In order to measure the attack space expansion due to the key sharing, [7] presented a large-scale study of key

sharing in today's HTTPS ecosystem.

Since the authentication mechanisms adopted in the current CDN architecture have shortcomings as described above, there have been existing works [3, 8, 9], which tried to solve the delegation problem without entrusting the hosting provider with the private key or the shared certificate. These works provide alternative ways to enhance the delegation issues in third party CDN authentication; however, the performance is not desirable in terms of network latency or practicality. In this paper, we will introduce another mechanism to solve the delegation problem of third-party edge networks like CDNs and compare our proposed solution with previous studies. The goal of our proposed mechanism is as follows:

- Avoid sharing credentials with the hosting providers.
- Provide efficiency in terms of latency.
- Ensure flexible deployment with lightweight operations.

To fulfill the goal mentioned above, we used an object called "cross credential" which can prove the relationship between the CDN edge and the origin server. We used the back-end communication between CDN edge server and the origin web server to periodically generate the CC which has validity duration. For verifying the generated CC at the client browser, we extended the standard TLS protocol used in the front-end communication between the CDN edge server and the client browser.

The remainder of the paper is organized with following chapters. Chapter 2 introduces the background of CDN and how it works to serve its customers. Chapter 3 describes the related works which attempted to solve the third party authentication issues in delegated edge networks similar to this study.

Chapter 4 explains the structure, design principles and implementation of the proposed scheme, *TLS-CC* in detail. Chapter 5 introduces the implementation and Chapter 6 evaluates the performance of the proposed scheme by comparing with the performance of previous studies. Chapter 7 finalizes the paper with the conclusions.

Chapter 2

Background

2.1 Content Delivery Network (CDN)

CDNs, which are third party hosting providers designated to improve the performance and scalability of web services, have become an important part of Internet infrastructure these days. By using multiple edge servers across geographical locations around the world, CDNs improve the delivery of static and media streaming contents. Such distributed architecture also provides some protection against DDoS attacks as well. To speed up page loading and maximize bandwidth, global content requests from end users are automatically routed to the nearest edge servers for the web services.

Web access using the CDN architecture is a two-step process as shown in Figure 2.1. The first step is taken at the front-end channel between the client and the edge server, which is also referred to as request routing [10]. Unless the user directly connects to the desired edge server by overriding

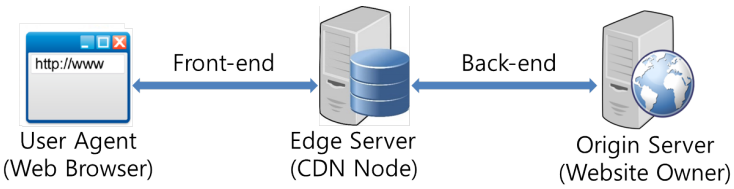


Figure 2.1: Web access using the CDN architecture

the CDN's edge selection as described in [11], the request is routed to the edge server which is geographically closest to the user. Most of the commonly used request routing techniques are URL rewriting and DNS-based request routing. The second step, called the request responding is taken at the back-end between the edge server and the origin web server. There are two types of mode in this step: push mode and pull mode [3]. In push mode, website owners upload their content to the CDN before the user requests the content. Pull mode is taken place when there is a "cache miss" at edge server. If a user requests the content which the edge server does not have, it must obtain the requested content from the origin web server to give response to the user. When the response is received from the origin web server, the edge server stores the content in its cache so that request of the same content can be served directly from the edge server next time.

2.2 SSL certificates for Edge Authentication

SSL certificate, which is an electronic document binding of a subject to a public key, stands for the online authentication in the HTTPS web services. A certificate generally contains information about the certificate owner, certificate usage, validity duration, and the certificate issuer who signs this in-

formation. Valid certificates are issued by Certificate Authorities (CAs), who maintain a list of all signed and revoked certificates. Unless the certificate is a self-signed certificate, there is a logical certificate chain from the root certificate to the leaf certificate through zero or more intermediate certificates. The most widely accepted standard for digital certificates is X.509 [12], which is commonly used in Internet security protocols (such as SSL/TLS).

When a user accesses a website via HTTPS (i.e., HTTP over SSL/TLS), the user's browser says hello to the web server, establishes the connection, and then receives the certificate of the server. On receiving it, the user agent checks the validity of the certificate by examining the chain of the leaf to the trusted root. Thus, to provide HTTPS web services using CDN architecture, the edge server needs to claim itself to be the origin server or prove its relationship with the origin server. There are two types of certificates currently used to support HTTPS connections using CDNs as described below.

2.2.1 Custom certificate

This is the case the origin server issues delegation to the CDN by uploading its private key and certificate which has the domain name of the origin server specified in the Common Name (CN) field. When a HTTPS request from a client is received, the CDN edge server sends this certificate and the client encrypts the pre-master secret using the public key of the certificate. To establish a secure channel on behalf of the origin server and provide the service, the edge server must decrypt the encrypted message sent by the client using the private key of the origin server. Despite its simplicity, this approach has the major drawback of sharing the private key as mentioned

in [3] because numerous edge servers of CDN need to hold the copy of origin server's private key and this may increase the attack surface.

2.2.2 Shared certificate

Unlike the custom certificate, this approach does not require the origin server to share her private key with the edge servers. Instead, the origin server need to allow the CDN to issue a certificate by putting its domain name in the Subject Alternative Name (SAN) field of the CDN's certificate. This method reduces the risk of private key sharing of the origin server; however, it also has drawbacks as mentioned in [3]. The main drawback is that there are usually many domain names in the SAN list of the certificate because a CDN typically supports many content providers. If the corresponding private key is compromised by an attacker, he can impersonate all of the domains without any warning in the user browser. In addition, the revocation process becomes complicated since the origin server must request the CDN to remove her domain name in the SAN list instead of asking her CA independently as in the custom certificate approach mentioned above.

Chapter 3

Related Works

3.1 DANE-based HTTPS Delegation

The first related study that analyze the problems of CDN certificates and introduce a solution is conducted by Jinjin Liang et al. [3]. In that paper, the authors introduced an extension of DNS-based Authentication of Named Entities (DANE) [13] to solve the problem of delegation in CDN edge networks. The concept of the solution is straightforward. Without relying upon the CA to authenticate the relationship between the certificate of the server and that of its domain name, this scheme use the DNS by securing the binding with Domain Name System Security Extension (DNSSEC) [14]. To issue a delegation, the origin server adds her certificate and that of her CDN as her TLSA records at the DNS server. When a user connects to the edge server and receives the certificate of the CDN, he further issues a DNS query to request the TLSA records of the origin server to see whether the edge is the authen-

ticated CDN node or not. If the TLSA record of the origin server includes the certificate of the CDN, then the edge is verified. Despite its simplicity, this solution has two major shortcomings. The first one is the deployment flexibility. Since this scheme depends upon the DANE and DNSSEC, the deployment will be complicated and might take time. The second one is the efficiency issue. The larger the size of the DNS response, the more resources is needed to cache the response.

3.2 CloudFlare's Keyless SSL

Another related work is Keyless SSL [4], which is proposed by one of the well-known CDN vendors, CloudFlare. Since the sharing of origin server's private key has the problem of increasing attack surface, the main purpose of the scheme is to move the private key operations of the edge server to a remote key server which is put on the origin server's side. When a request from a user reaches the edge server, it will do the TLS handshake with the origin server certificate. But the edge server does not hold the private key of the origin server, so the private key operations like decrypting the pre-master secret or verifying the signature on DH parameters are moved to a remote key server which is put on the customer's infrastructure. This technique has the advantage that the origin server can retain control of her private key. However, this scheme also has a shortcoming which is the need of long round-trip time to the key server during the TLS handshake to get the decrypted pre-master secret. According to the measurement of the TLS handshake time mentioned in [4], we can see that the adoption of Keyless SSL at the CDN edge nodes is 100 ms shorter than the direct handshake with the origin server,

which only improved 20% of the TLS handshake time.

3.3 HTTPS-based Redirection for Delegation

Another approach is [9], which uses the HTTPS-based redirection with a token from the origin to the CDN edge server. The special security token called 'trTOKEN' (tls redirection TOKEN) binds together the certificates of the origin server and the CDN. At the end of the TLS handshake with the client, the origin server sends a redirection URI with the trTOKEN to indicate the trusted delegation to its CDN. This method solves the delegation problem of edge networks; however, due to the need for two TLS handshakes, there is a significant increase in network latency.

Chapter 4

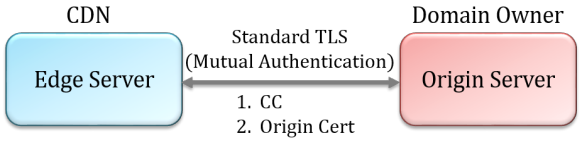
TLS Cross Credential (TLS-CC)

4.1 Design Principles

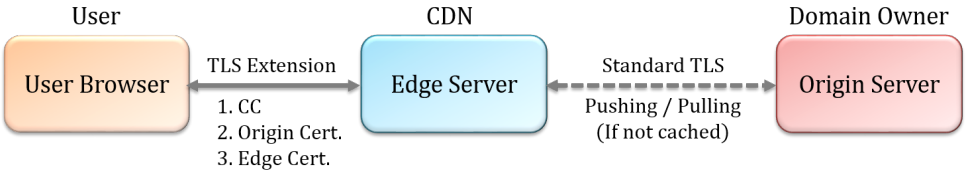
The basic design principles of our proposed scheme is as follows:

- No credential sharing: Sharing of the private keys or certificates should be avoided since it may increase the attack surface.
- Efficiency: No access to origin server during handshake unless necessary.
- Revocation Convenience: The revocation of one's own certificate (or private key) must be independent of another entity.
- Deployment Flexibility: The deployment must be easy and manageable.

Figure 4.1 shows the design overview of our proposed scheme. The main idea of the scheme is to make the user recognize two separate entities (i.e. the CDN and the origin server), and figure out their relation. Considering the design principles mentioned above, we model an object called Cross Credential



(a) Back-end channel



(b) Front-end channel

Figure 4.1: Design Overview of Proposed scheme

(CC), which binds the relationship between the CDN and the origin server. The generation of CC can be performed periodically between the origin and the CDN edge server at the back-end channel, so that the edge server can avoid access to the origin server during the handshake with the user. For the verification of the generated CC at the front-end channel, we put the CC into a `ServerHello` message of the edge server with the use of TLS extension. The extended TLS is called TLS-CC.

4.2 Cross Credential (CC)

Figure 4.2 shows the structure of Cross Credential (CC), which is a cryptographic proof representing the binding between the CDN and the origin server. The role of each component of the CC can be described as follows (Table 4.1 includes the definitions for the symbols in the equation):

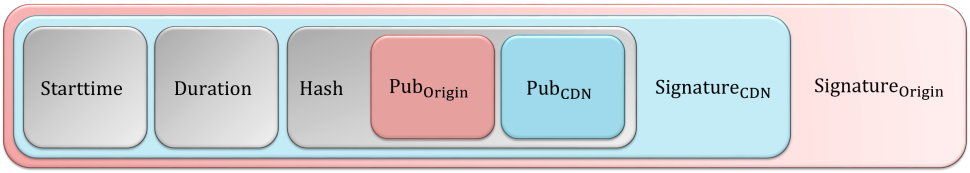


Figure 4.2: The structure of Cross Credential (CC)

$$CC = Sig_{Origin}(Sig_{CDN}(t_s||t_d||H\,(Pub_{Origin}||Pub_{CDN})))$$

$H\,(Pub_{Origin}\|\|Pub_{CDN})$	Prove the relation between the origin server and CDN
t_d	Describe the validity duration of the CC
t_s	Describe the starting time that the CC is generated
Sig_{CDN}	Signature of the CDN for the agreement of CC
Sig_{Origin}	Signature of the origin server for the agreement of CC

Table 4.1: Components of Cross Credential (CC)

The core component of the CC is the hash value which is generated by the concatenation of the public keys of origin server and its associated CDN. This represents that the owner of the first public key (i.e. the origin server) delegates its service to the owner of the second public key (i.e. the CDN). Thus, when the hash value of the CC is checked for the CC verification at the front-end, the user can be aware of the delegated relationship between the origin and the edge server. The second important component of the CC is the validity duration (t_d) which is defined as the interval from the start time (t_s) to the sum of the start time and the duration. The main purpose of validity duration is for the revocation convenience. If there is no validity duration, the generated CC can be used permanently as long as either of the

entities revokes its own private key, which is an expensive procedure.

4.2.1 CC Generation

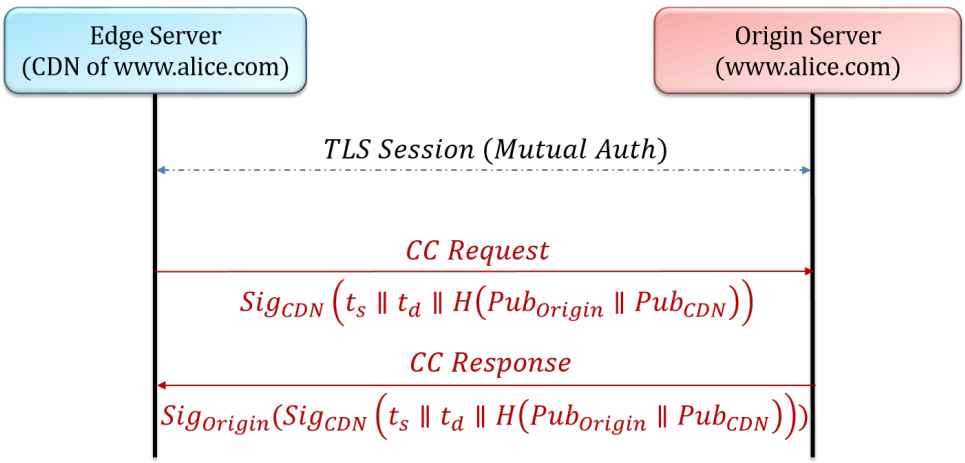


Figure 4.3: Generation of Cross Credential (CC)

The generation of the CC is done at the back-end as shown in Figure 4.3. Once the edge and the origin server have established the TLS session via mutual authentication, the edge server sends the CC request by putting its signature as an indication of the agreement of contents inside the CC. Note that during the handshaking, Origin Server must check whether CDN Edge Node is valid by white-listing with the domain name, IP addresses, or using shared secret on the contract. When the CC request from the edge server is received, the origin server checks the components of the request and issue the CC by putting its signature as an indication of the agreement of request that the edge sends. Note that the validity period is the duration of the CC, so it

should be set as short as possible. For that purpose, we can design the CC generation to be performed automatically and periodically, so that it can be efficiently updated without human intervention.

4.2.2 CC Verification

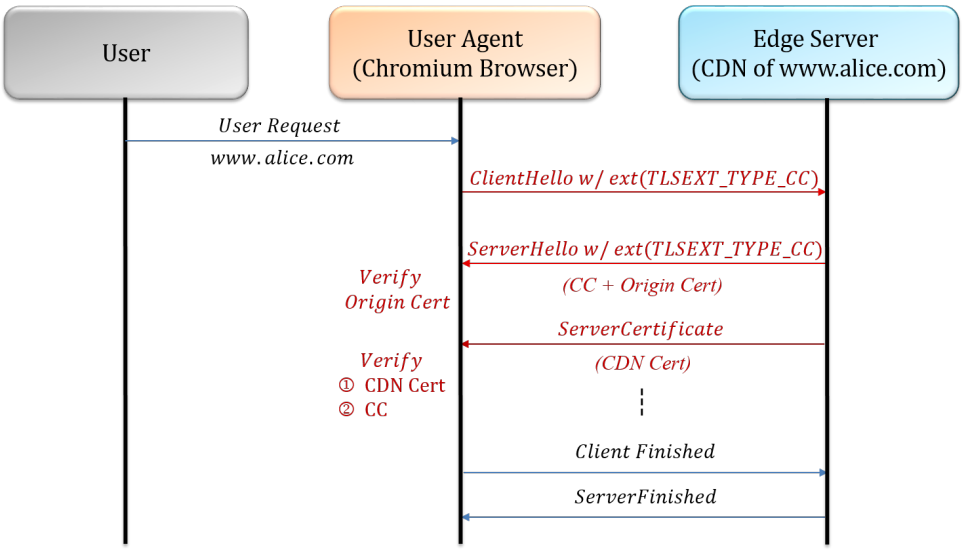


Figure 4.4: Verification of Cross Credential (CC)

The verification of the CC for the authentication of edge server is performed at the front-end as shown in Figure 4.4. When the user agent (for example, chromium browser) makes a request for a domain handled by a CDN, the request is routed to the physically closest CDN edge server according to the request routing techniques defined in [10]. To serve the user instead of the origin domain owner, the edge server needs to authenticate itself. For that, we use the TLS-CC, which is an extension of the standard TLS protocol.

We adopt the concept of TLS extension described in [2] to extend the TLS handshake by inserting the CC in `ClientHello` and `ServerHello` messages. We define a new extension type called TLSEXT_TYPE_CC as an indicator of our message. Using the TLSEXT_TYPE_CC with zero content length (cc_len), the user agent sends `ClientHello` to the edge server by expressing her intention to use TLS-CC if available. When the message is received, the edge server sends the `ServerHello` using the same type (i.e. TLSEXT_TYPE_CC) by inserting the CC and the certificate of the origin server in the extended message. Also, the edge server sends its own certificate in the `ServerCertificiate` message of the TLS handshake. Thus, upon receiving those messages, the user agent can process the certificate chain validations of the origin and edge server using the public keys of the two entities. After the verification of the certificates, the client checks the contents of CC as described in Session 4.2. If any of the verification procedure is failed, the client breaks down the connection. Otherwise, the client trusts the delegation of the origin to edge server and establishes the TLS connection by completing the remaining part of handshake messages.

Chapter 5

Implementation

We implemented our proposed scheme using the back-end channel to create the Cross Credential (CC) object mentioned in Session 4.2, and the front-end channel to verify it. Considering the deployment flexibility, we made the major changes on SSL libraries. We then linked these modified SSL libraries with the popular Nginx web server and the Chromium web browser to measure the performance and practicality of our proposed scheme.

5.1 User-side modifications

For a realistic performance evaluation, we modified the boringSSL, which is a customized version of OpenSSL for Google, in Chromium version 57.0.2984.0. Considering the compatibility with the existing SSL libraries, we extended the standard TLS protocol by defining a new extension type called TLSEXT_TYPE_CC that can indicate the intent of the user browser to use

TLS-CC if available at the edge server. By parsing the `ServerHello` from the edge server, the user browser can know whether the server supports the TLS-CC or not. If it is supported, the flag for TLS-CC called *cc_enabled* is set to '1' to perform the operations for CC verification as explained in Session 4.2.2. Otherwise, the *cc_enabled* flag remains as the default value '0' and the verification processes will be skipped. If any of the process during the CC verification is failed, the user browser will immediately break down the connection with the edge server.

5.2 Edge-side modifications

Since the edge server stands between the origin server and the user browser, we modified it to generate the CC along with the origin server and then communicate with the user browser to perform the verification. For that, we used the OpenSSL 1.0.2k and combined it with Nginx 1.10.1.

As mentioned in Session 4.2.1, the edge server generates the CC by establishing the TLS session using the mutual authentication with origin server at the back-end. Once the CC is generated, it is ready to use for the edge server authentication at client browser at the front-end. For such front-end communication, we modified the OpenSSL libraries to support the TLS-CC extension, and then built the Nginx web server with our modified OpenSSL source.

Similar to the concept of user-side modifications, when the `ClientHello` message is received from the user browser, the edge server checks the extension type of the message. If the type is TLSEXT_TYPE_CC, it sends the CC and the origin's certificate as the `ServerHello` extension message as men-

tioned in Session 4.2.2. It also sends the certificate of itself as well so that the user browser can verify the CC and authenticate the edge server using the public keys from the two certificates.

Chapter 6

Evaluation

6.1 Experiment Setup

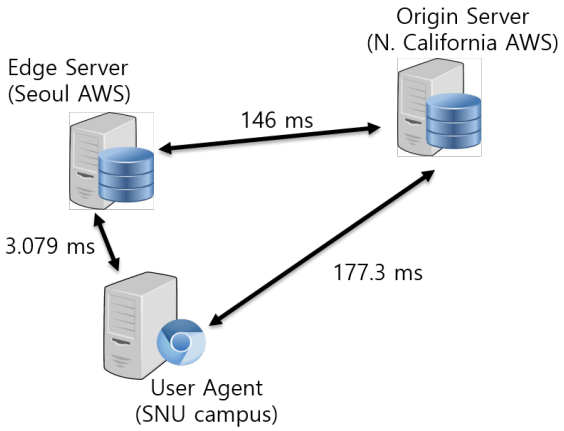


Figure 6.1: Experiment Setup Overview

The experiment setup to evaluate our proposed TLS-CC scheme is shown in Figure 6.1. For the edge and origin server, we use Amazon EC2 (Elas-

tic Compute Cloud) instances located at Seoul and North California. Both instances are Ubuntu 16.04 with Intel Xeon CPU E5-2676 @2.40GHz and 1GB memory. Round Trip Time (RTT) measured by the ping time between each of the instance is also mentioned in the figure. For the client, we use a computer which is located at our Seoul National University (SNU) campus. All the servers are running Nginx web servers built together with OpenSSL and the client is trying to connect via the Chromium browser.

6.2 Client-side Evaluation

Two experiments were taken at the client side to evaluate the performance of our proposed TLS-CC scheme. The first experiment measured the TLS handshake time of different delegation techniques, and the second experiment used different signature algorithms called RSA2048, RSA4096, and ECC256 to compare the TLS handshake of our proposed scheme and the custom CDN scheme (i.e. the one with the custom certificate).

6.2.1 Comparison of different delegation schemes

In this experiment, we measured the TLS handshake time of different delegation schemes as described in Table 6.1. To compare the TLS handshake time of four different delegation schemes (i.e. CDN (Custom), TLS-CC, HTTPS Redirection, Keyless SSL), we used the origin custom scheme as the baseline of the measurements. For each measurement, we use RSA 2048 bits for the key pairs of origin and edge server, and headless chromium client as mentioned in [15] to send request to the edge or origin server 100 times.

Origin (Custom)	Direct handshake with the origin server
CDN (Custom)	Handshake with the edge server using the custom certificate
TLS-CC	Handshake with the edge server using our proposed scheme
HTTPS Redirection	Handshake first with the origin server which is then redirected to the edge server
Keyless SSL	Handshake with the edge server whose private key operations are done at the origin's key server

Table 6.1: Overview of CDN delegation schemes

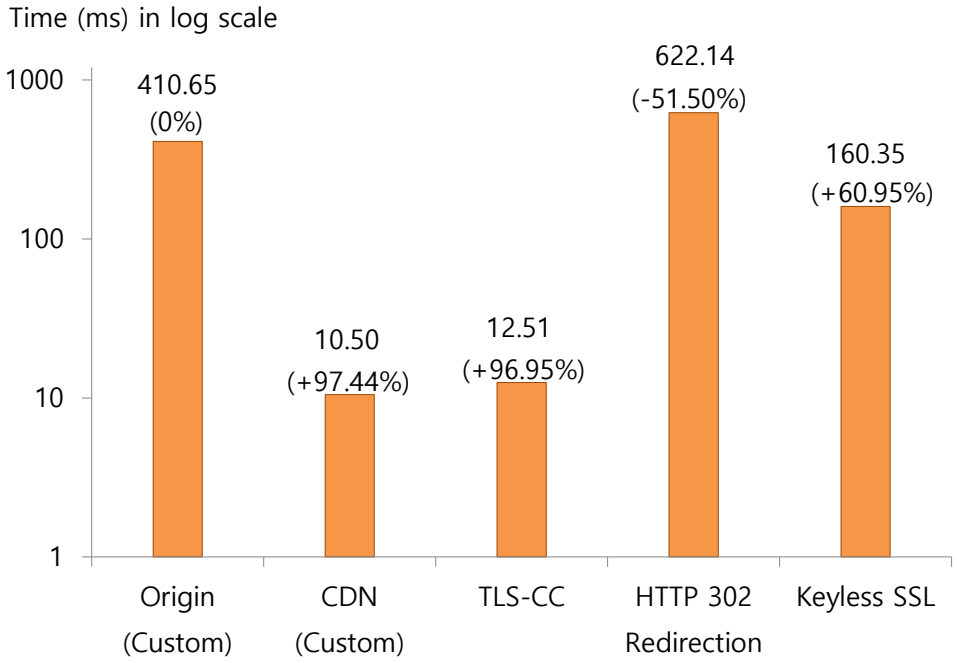


Figure 6.2: TLS handshake time of CDN delegation schemes

Figure 6.2 shows the averages of the TLS handshake time in milliseconds (ms) for each of the delegation scheme. In order to compare the performance of each scheme in terms of latency, we calculated the percentage of gain with reference to the baseline Origin (Custom) scheme defined as follows:

$$\frac{TLS\ handshake\ time\ (Origin) - TLS\ handshake\ time\ (Scheme)}{TLS\ handshake\ time\ (Origin)} \times 100(\%)$$

Our proposed TLS-CC scheme shows 96.95% gain while the CDN (Custom), HTTPS Redirection and Keyless SSL shows 97.44%, -51.5% and 60.95% respectively. Since the TLS handshake delay of our scheme is almost the same as the CDN (Custom), we can say that our scheme has efficiently solved the problem of credential sharing with this slight delay of 2 ms. We can also note that HTTPS redirection scheme got minus gain due to the need of two handshakes (i.e. first with the edge server and then redirect to the origin server), and Keyless SSL also achieved less gain than ours due to the single round trip to origin server for the private key operations.

6.2.2 Comparison of TLS-CC and CDN Custom scheme

Note that CDN custom scheme needs only one certificate which is of edge server, and our proposed scheme needs two certificates which is of both edge and origin server. So, in order to evaluate the computation overhead of the origin server's certificate chain validation and the CC verification of our proposed scheme, we also measured the TLS handshake time of our scheme and that of the CDN (Custom) using different signature algorithms. The three cases of signature algorithms for our measurement is listed in Table 6.2.

CDN Custom scheme	Proposed TLS-CC scheme
RSA 2048 (edge's certificate)	RSA 2048 (origin's certificate) + RSA 2048 (edge's certificate)
RSA 4096 (edge's certificate)	RSA 4096 (origin's certificate) + RSA 4096 (edge's certificate)
ECC 256 (edge's certificate)	ECC 256 (origin's certificate) + ECC 256 (edge's certificate)

Table 6.2: Signature algorithms for the experiments

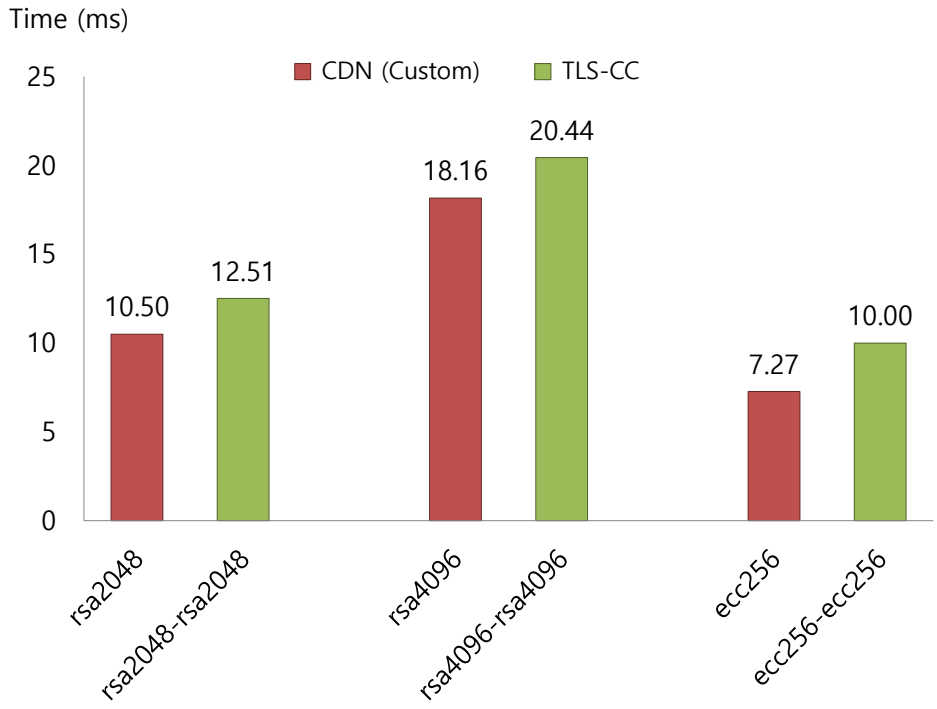


Figure 6.3: TLS handshake time of CDN Custom and TLS-CC for different signature algorithms

Similar to the experiment we did in Session 6.2.1, we sent request to the edge server 100 times from the headless chromium client. The result of the experiment is shown in Figure 6.3. The TLS handshake delay of our TLS-CC is again 2 ms longer than that of the CDN (Custom) scheme for each key pair due to the verification of one more certificate (i.e. origin certificate) and the CC. However, with the cost of slight delay, our scheme has solved the problem of credential sharing in CDN edge networks.

6.3 Server-side Evaluation

To evaluate the performance and practicality of our proposed TLS-CC scheme, we did two experiments at the server side. In the first experiment, we analyzed the additional bytes required to send for the edge verification of our proposed scheme. In the second experiment, we compared the memory utilization of the CDN custom scheme and our TLS-CC.

6.3.1 Comparison of outgoing traffic at Edge Server

Since our proposed TLS-CC scheme sends additional data for the CC verification as mentioned in Session 4.2.2, we measured the total bytes sent from the edge server during the TLS handshake for the traditional TLS and our TLS-CC. Similar to the client-side experiment, we use the same combinations of key pairs as described in Table 6.2.

The result of our experiment is shown in Figure 6.4. With our TLS-CC scheme, 1,888 bytes, 2,073 bytes and 1,019 bytes are additionally required to send from the edge server for RSA2048, RSA4096, and ECC256 signature algorithms. We can also note that the overhead of the origin certificate is

the biggest among the overhead bytes in each of the cases. Based on our observation in the client-side TLS handshake measurements, we have seen that this overhead did not incur much delay during the TLS handshake with the client. So, we believe that this impact of increasing traffic is negligible with fast network speeds which is more dominant for the latency.

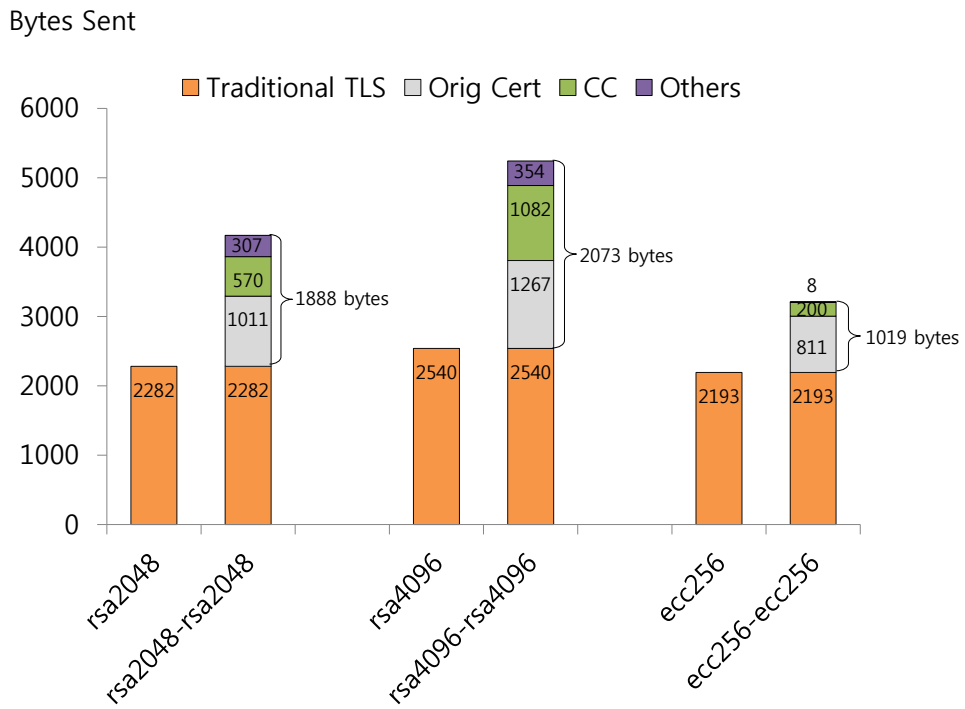


Figure 6.4: Outgoing traffic of traditional TLS and TLS-CC for different signature algorithms

6.3.2 Comparison of memory utilization at Edge Server

In this experiment, we measured the overhead of the CC at the edge server-side in terms of the memory utilization. We did our evaluation by sending

different number of client requests per second and checked the memory footprint every second. The result of our measurement is shown in Figure 6.5. The x-axis is requests per seconds we sent and the y-axis is the percentage of memory used for total memory (1 GB). We have noticed that the memory utilization of both schemes increases linearly as the request rate increase. The memory utilization of our TLS-CC is 4.2 times that of the TLS custom scheme due to the overhead of the CC and one more certificate (i.e. the origin server's certificate).

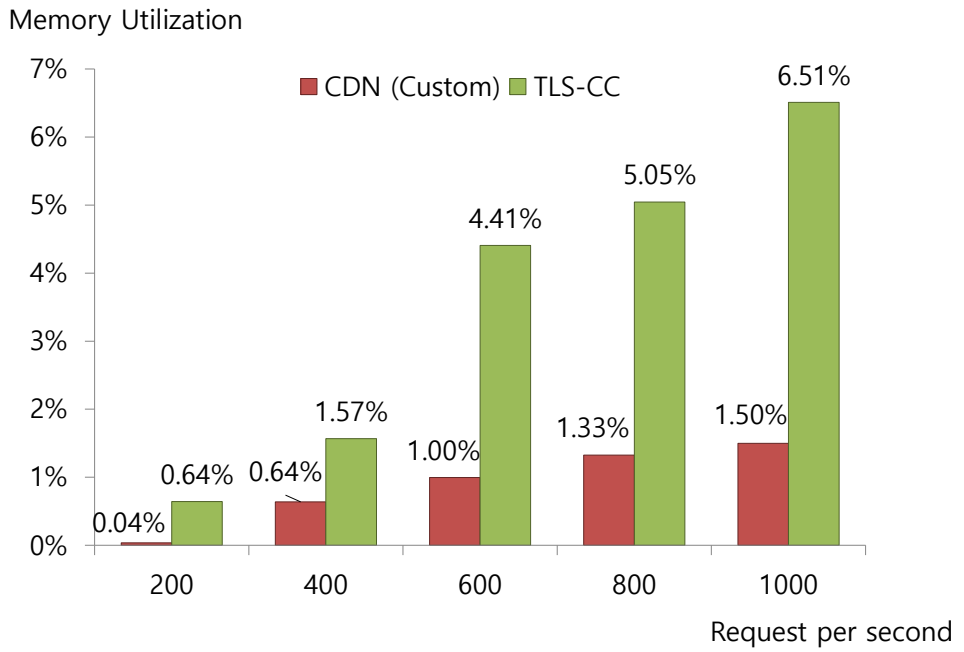


Figure 6.5: Memory Utilization of CDN Custom and TLS-CC

Evaluation Metrics	Custom (CDN)	TLS-CC (Proposed)	HTTP 302 Redirection	Keyless SSL
Attack Surface (Origin Key)	Big (#Edges)	Small (#Origin)	Small (#Origin)	Small (#Origin)
Stolen CDN Key	N/A	Safe	Not Safe	N/A
Delegation Revocation Time	Long (Cert. Revocation)	Short (CC Duration)	Immediate	Immediate

Table 6.3: Security comparison of CDN delegation schemes

6.4 Security Evaluation

Table 6.3 shows the security properties of each CDN delegation schemes. *Attack Surface (Origin Key)* denotes the number of servers holding the private key of the origin server. The more servers with the origin's private key, the greater the likelihood of attack. Therefore, the attack surface of the CDN (Custom) is the largest among the delegation schemes since all CDN edge nodes hold the private key of the origin server. *Stolen CDN Key* denotes the impact of a stolen CDN private key. Custom (CDN) and Keyless SSL are not relevant because they do not use CDN's private key. HTTP 302 Redirection scheme is not safe when the private key of CDN is stolen since the origin server will pass the url of the CDN to the user and the attacker can launch a phishing attack. Proposed TLS-CC scheme is safe since the origin server will make sure the CC is only issued to the valid CDN edge node as described in Session 4.2.1. *Delegation Revocation Time* denotes the time required for the revocation of the delegated relationship between the origin server and

the CDN. If the revocation of the relationship is not immediately effected, the malicious or compromised CDN can use the time window for malicious purpose. Custom (CDN) has relatively long delegation revocation time since the origin server need to issue a certificate revocation request to its CA, and it might take some time depending on the type of certificate used. Especially, if the origin server's certificate is an Extended Validated (EV) type, the revocation process can be quite expensive and time consuming [3]. In the case of our proposed TLS-CC scheme, the delegation revocation time can be adjusted according to the need since it is determined by the validity period of the CC.

Chapter 7

Conclusion

CDNs play an important role in today's internet infrastructure because they provide a faster experience to the end-users and protect against DDoS attacks. However, despite its convenience, there are some security issues regarding to the delegated third party authentication. In particular, to provide HTTPS services over CDN edge networks, there is an authentication problem due to the credential sharing among delegated edge networks using custom or shared certificates. There are several previous works that have introduced solutions to this problem; however, solving it in an efficient way is still challenging.

To tackle such challenge, we propose to use a cryptographic object called Cross Credential (CC), which represents the binding between the CDN and the origin server. Since the main idea of our scheme is to let the users recognize and relate the two separate entities (i.e. the CDN and the origin server) for the delegated authentication, we have efficiently solved the problem of

credential sharing.

To evaluate the performance and usability of our TLS-CC scheme, we conducted experiments on both client and server side. Compared to other previous solutions, our proposed scheme has reduced the delay of the TLS connection setup significantly, but with some level of memory overhead due to the verification of one more certificate and the CC. However, for the secure HTTPS operations in the CDN edge networks, we believe that the authentication scheme should express the relationship between the CDN and the origin server, and we disclose some guidelines for future research.

Bibliography

[1] W. Scott, T. Anderson, T. Kohno and A. Krishnamurthy, "Satellite: Joint analysis of CDNs and network-level interference," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, USENIX Association, June 2016. pp. 195-208.

[2] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, August 2008.

[3] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan and J. Wu, "When HTTPS meets CDN: A Case of Authentication in Delegated Service," in *Security and privacy (sp)*, 2014 ieee symposium on. IEEE, May 2014. pp. 67-82.

[4] D. Stebila and N. Sullivan, "An analysis of tls handshake proxying," in *Trustcom/BigDataSE/ISPA*, 2015 IEEE. Vol. 1. IEEE, August 2015. pp. 279-286.

[5] M. Myers, R. Ankney, A. Malpani, S. Galperin and C. Adams, "RFC 2560, X. 509 Internet Public Key Infrastructure Online Certificate Status Protocol-OCSP," Internet Engineering Task Force RFC, June 1999.

[6] H. Takabi, J. B. Joshi and A. Gali-Joon, "Security and privacy challenges in cloud computing environments," in *IEEE Security & Privacy*, Volume: 8, Issue: 6, December 2010. pp. 24-31.

[7] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove and C. Wilson, "Measurement and analysis of private key sharing in the https ecosystem," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, October 2016. pp. 628-640.

[8] N. Sullivan, "Keyless SSL: The Nitty Gritty Technical Details," https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/

[9] S. Slovetskiy, "Approaches to HTTPS-based Request Routing and Delegation," RFC Internet-Draft, July 2015.

[10] A. Barbir, B. Cain, R. Nair and O. Spatscheck, "Known Content Network (CN) Request-Routing Mechanisms," IETF RFC 3568, July 2003.

[11] S. Triukose, Z. Al-Qudah and M. Rabinovich, "Content Delivery Networks: Protection or Threat?," in *European Symposium on Research in Computer Security*, Springer Berlin Heidelberg, September 2009. pp. 371-389.

[12] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, IETF, May 2008.

[13] J. Schlyter and P. Hoffman. The DNS-based Authentication of Named Entities (DANE) Transport Layer Security (TLS) protocol: TLSA RFC6698, August 2012.

[14] M. Larson, D. Massey, S. Rose, R. Arends and R. Austein. DNS security introduction and requirements. RFC 4033, March 2005.

[15] Headless Chromium, https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md

초록

최근 미디어 및 엔터테인먼트 회사와 같은 대부분의 콘텐츠 제공 업체는 CDN (Content Delivery Network) 서비스를 사용하여 보다 빠른 응답속도와 높은 가용성을 제공한다. 전 세계적으로 분산된 서버 인프라를 사용하여 네트워크 트래픽을 처리하는 CDN은 최종 사용자에게 보다 빠른 체감속도를 제공하고 DDoS (Distributed Denial of Service) 공격으로부터 어느 정도의 보호를 제공한다. 그러나 이러한 이점에도 불구하고 CDN에는 제 3자 에지 네트워크 인증과 관련된 몇 가지 단점이 존재한다. 현재의 메커니즘은 CDN 업체에게 개인 키를 맡기거나 인증 기관이 CDN에 공유인증서를 발급하도록 허용한다. 두 메커니즘은 개인 키 공유로 인한 공격 가능성 확장 또는 도메인 혼선 및 복잡한 인증서 폐기 프로세스의 측면에서 단점을 갖는다.

본 논문에서는 CDN 에지 네트워크에서 CDN을 신뢰하거나 인증 기관이 CDN에 공유인증서를 발급할 필요가 없는 인증 메커니즘을 제안한다. 본 논문에서 제안하는 메커니즘은 CDN 에지 서버와 원 서버 간의 위임된 관계를 증명할 수 있는 "상호 인증(CC:Cross Credential)"을 사용하여 현존하는 솔루션에 비해 지연 시간과 성능 오버헤드를 크게 개선할 수 있다. 제안하는 메카니즘은 백-엔드에서 CC를 만들고 프런트-엔드에서 표준 TLS (Transport Layer Security) 프로토콜을 확장하여 CC를 확인하는 방식으로 구현하였다.

주요어: Content Delivery Network, Transport Layer Security, 위임, 인증

학번: 2015-23302